1 Herkunft und Eigenschaften von Linux

1.1 Einleitung

Eigenschaften:

- echtes 32-bit Betriebssystem
- UNIX-ähnlich
- POSIX-konform
- enthält keinen Original-UNIX-Code

Anfangs der 70^{er} entstand das 1. UNIX in den AT&T Bell-Laboratories; Ziel war relativ kleine Maschinen effektiv zu nützen; multi-user-Betrieb; Ende der 70^{er} gab es 3 UNIX-Hauptlinien:

- AT&T-UNIX
- PC-UNIX (Microsoft-Lizenz)
- BSD-UNIX (University of Berkeley)

Nachdem die UNIX-Lizenzen drastisch verteuert wurden entstanden UNIX-look-alike Systeme z.B.: Coherent, MINIX von J. Tanenbaum (microkernel-System). Krieg der verschiedenen UNIX-Systeme; Standardisierungsversuche; POSIX-Standard (Portable Operating System for Computer Environments) von IEEE; Benutzerschnittstelle: X-Windows ist das kostenlose Kernsystem worauf unterschiedliche Oberflächen aufbauen;

1.2 Aufgaben eines Betriebssystems

Shells wandeln die Benutzerbefehle in die entsprechenden Aufrufe von Systemdiensten um.

Technische Voraussetzungen

Interruptsystem

Timer Interrupt Vermeidung, daß 1 Anwendungsprogramm alle anderen Anwendungsprogramme und das Betriebssystem blockiert;

Privilegienstufen ab Intel 386 gibt es 4 Privilegienstufen "Ringe" wobei die höchste Prioritätsstufe Nr. 0, der Kernel-Modus ist (z.B.: Speicheroperationen, Gerätezugriff);

Virtuelle Speicherverwaltung der von einem Anwendungsprogramm benötigte Speicherplatz muß im Hauptspeicher nicht zusammenhängend sein;
Speicherauslagerungen:

Swapping ganze Programme werden auf die HD ausgelagert;

Paging Teile eines Programms werden auf die HD ausgelagert;

Die LINUX-Lösung

Resource Rechenzeit: Jedes aktive Programm wird in einem Prozeß ausgeführt, der relativ zu den anderen Prozessen eine gewisse Priorität hat ⇒ Warteschlange. Ein *Alterungszähler* erhöht im Laufe der Zeit die Priorität eines wartenden Prozesses, sodaß er nicht ewig von höherprioren überholt wird. Mittels der *Zeitscheibe* wird überwacht, daß jeder Anwendungs- oder Betriebssystem-Prozeß nur eine maximal zulässige Zeit ungestört die CPU in Beschlag nimmt, nach Ablauf dieser Zeitscheibe wird überprüft welcher Prozeß als nächster dran kommt. Läuft ein Prozeß gerade auf der CPU so ist sein Status: Running, ist er in der Warteschlange, so ist sein Status: Active.

Speicherverwaltung: Jede Operation, die in die Hauptspeicherverwaltung eingreift (z.B.: in einen Speicherbereich adressieren) ist ein privilegierter Befehl ⇒ Aktivierung des Betriebssystems. Bei einem illegalen Zugriff wird der schuldige Prozeß abgebrochen, eine Fehlermeldung "bus error" oder "memory fault" abgesetzt und der Hauptspeicherausdruck in eine Datei namens core geschrieben.

1.3 Prozesse

Vereinfacht besteht ein Prozeß aus:

- Verwaltungsdaten
- dynamischer Bereich: Speicherbereich für Laufzeitstack;
  dynamische Speicherverwaltung des Programms auf heap;
- Programm-code und -Daten (statischer Teil eines Prozesses)

Prozeßbaum

Prozeß Nr. 0 hat als einziger keinen Vorgängerprozeß; er entsteht beim booten;
Prozeß Nr. 1 – Init-Prozeß:

- Start gewisser Systemdienste gemäß /etc/inittab und der rc-Dateien im Verzeichnis /etc;
- Systemanmeldung (über Konsole oder serielle Leitung) mit dem Getty-Prozeß;
- Anmeldeshell;

Bei Beenden des Anmeldeprozesses werden, falls man dagegen keine Vorkehrungen getroffen hat, alle Kind- und Enkelprozesse mit-beendet. Danach startet init einen neuen Getty-Prozeß. Die Prozeßhierarchie wird unter dem Verzeichnis /proc verwaltet.

Waisen Wird ein (Eltern-) Prozeß, der noch von ihm abstammende(Kind-) Prozesse laufen hat, beendet, so werden diese Kindprozesse ⇒ Waisen vom Prozeß Nr. 1 adoptiert.

Zombies Jeder Prozeß liefert bei seiner Beendigung noch einige Daten an seinen Elternprozeß. Es werden zwar der Hauptspeicher und ein Großteil der Verwaltungstabellen freigegeben, aber nicht alle, bis das der Eltern- oder Adoptivprozeß erledigt. In der Zwischenzeit existiert der Prozeß als Zombie (Prozeßstatus Z nach Aufruf der Prozeßtabelle mit ps).

Dämonen Systemprozesse, die im Hintergrund laufen, d.h. sie haben keinen Ausgabeterminal (z.B.: Init-, Drucker-, Netzwerkprozesse) (Terminalangabe ? bei Aufruf der Prozeßtabelle mit ps).

1.4 Dateisystem

Festplatte vor der Installation von Linux in virtuelle Laufwerke und die swap-partition von Linux (eigenes Format) partitionieren.

Linux unterstützt viele verschiedene Dateiformate (z.B.: ext2, MSDOS, ISO 9660 (CD-Roms)).

Das jeweilige Dateisystem mit mkfs auf die jeweilige Partition bringen.

Aufbau eines UNIX-/Linux-Dateisystems

bootblock enthält den boot-loader zumindest wenn es die Wurzelpartition ist;

superblock enthält Infos über Partition und Dateisystem; beim mounten liest Linux den superblock; Änderungen werden bis zum unmounten (umount), sync oder Herunterfahren des Betriebssystems nur in der Kopie im Hauptspeicher gemacht;

Inode-Liste enthält Infos über Dateien;

Datenblöcke enthalten die Daten selber und die Verweise von den Inodes zu ihnen;

Dateihierarchie

Die Dateihierarchie ist baumförmig abgesehen von den Links (Sprünge quer durch den Baum);

Verzeichnis /proc: Prozeßverwaltung: Die Inodes in diesem Dateibaum werden zur Verwaltung der Prozeßhierarchie und der zu den Prozessen gehörenden Speicherbereiche und Verwaltungsinformationen genutzt.

Verzeichnis /dev: Einträge der Geräte und Resourcen;
ls -l gibt bei der

- Typangabe an:

  b blockorientiertes Gerät

  c zeichenorientiertes Gerät

- statt der Dateigröße die device number (major-device-nr., minor-device-nr.) an:

  major-device-nr. Gerätetreiber

  minor-device-nr. I/O-port, spezielle Funktionalität, ...

Inode

wichtigste Verwaltungsstruktur für Dateien; 64 Bytes groß;
2 Teile:

Informationsteil Inhalt: Kennung des users, der group, Rechtemaske, Dateigröße, Dateityp, verschiedene Zeitmarken, Link-count;

Verweisteil
- Adressen der zur Datei gehörenden Datenblocks (indirekt adressiert);
- bei Geräteknoten: major-device-nr., minor-device-nr.;

Das Programm fsck versucht beim Wiederhochfahren nach einem Stromausfall bzw. Systemabsturz die Inode-Einträge und Superblöcke wiederherzustellen (Datenverlust möglich);

2 Installation von Linux

fdisk Platte partitionieren;

mkfs als Linux-Partition formatieren;

zusätzliche Festplatten partitionieren, formatieren und Einträge in die Datei /etc/fstab (Gerätenamen, Anbindungspunkt im Verzeichnissystem, Art der Einbindung);

df –hT (diplay filesystem); Anzeige der Art des filesystems der Größe, der Größe des noch freien Platzes und des mount points.

du (disk usage); Zeigt die Größe des Arguments und seiner subdirektories in kB an.

fsck (filesystem check) Es können entweder die device names oder die mount points angegeben werden.

Erstellen einer Bootdiskette cat \vmlinuz > \dev\fd0

3 Tour durch Linux

Virtuelle Konsolen: hin- und herschalten zwischen ihnen mit Alt–Fx oder Strg–Alt–Fx, Fx $\in$ [F1, ..., F12];

.bash_profile: Diese Datei enthält die Anweisungen zur Konfiguration (Voreinstellung) der Shell;

Shell

In die Shell eingebaute Kommandos werden direkt augeführt, sonst wird in einer Variable PATH, nach den möglichen Pfaden der externen Kommandos nachgesehen; externe Kommandos laufen jeweils in einem eigenen von der Shell gestarteten Prozeß;

/etc/passwd – Benutzerdatenbank

Diese Datei stellt die Benutzerdatenbank dar; Aufbau einer Zeile dieser Datei: username: verschlüsseltes password: UID: GID: Kommentar: home_dir: login_program

UID user identification number; UID 0 ist dem superuser vorbehalten;

GID group-ID; Zugehörigkeit zu einer in der Gruppendatei definierten Gruppe;
Aufbau einer Zeile der Gruppendatei /etc/group:
groupname: leere Spalte; ursprünglich für Gruppenkennwort vorgesehen: Mitgliedsliste (Mitglieder durch Kommas getrennt)

newgroup Befehl zum Ändern der Gruppenzugehörigkeit;

passwd Befehl zum Ändern des eigenen Passworts; wärend dieses Programmlaufes wird dem Benutzer mithilfe des Set-UID-Bits für die Dauer des Programmlaufes praktisch das superuser-Recht zugeteilt ⇒ er kann sein eigenes Passwort ändern;

Kommentar enthält meist Raumnummer, Tel.Nr. bzw. Klappe;

home_dir Verzeichnis in dem sich der user nach dem login automatisch befindet; dieser Eintrag ist unbedingt erforderlich;

Identitätswechsel mit su

remote als superuser einloggen ist nicht erlaubt;

su *new_username* (substitute user): Befehl um die eigene Identität zu wechseln;

exit Befehl um wieder zur ursprünglichen Identität zurückzuwechseln;

Für schizofrene:

id gibt eigene UID, GID und mögliche Gruppenzugehörigkeiten aus;

who am i Ausgabe: hostname! username terminal login_date login_time;

Abmelden

exit wartet bis zuerst die Kindprozesse beendet sind;

logout beendet automatisch die Kindprozesse;

Strg–D beendet automatisch die Kindprozesse;

3.1 Wandern im Dateibaum

cd *Pfad* wechseln in das durch den Pfad bezeichnete Verzeichnis;

cd wechseln in das durch die Variable HOME bezeichnete Verzeichnis;

Zugriff auf externe Datenträger

mount ohne Argument ⇒ Liste der aktuell angehängten Geräte; der Befehl mount ist oft nur dem superuser erlaubt;

mount –**t***Typangabe Geräteknoten Verzeichnis (in das eingehängt wird)* z.B.: mount –tiso 9660 /dev/mcd /cdrom

Typangabe: ISO 9660 Dateisystem (CDRom-Norm);

Geräteknoten des CD-Rom-Laufwerkes: /dev/mcd;

Verzeichnis in das der Dateibaum der CD-Rom-Daten eingehängt wird: /cdrom;

umount Befehl zum Abhängen eines Laufwerkes;

Die Datei /etc/fstab enthält die eingehängten Geräte und ihre Dateibäume;

Wichtig bei Verwendung von Disketten: Reihenfolge: Diskette einlegen – mounten – bearbeiten – unmounten – Diskette entfernen einhalten; erst beim unmounten werden die Daten auf die Diskette geschrieben!

3.2 Dateien und Verzeichnisse bearbeiten

Verzeichniskommandos

rm (remove - entfernen) Löschen von Dateien, Verzeichnissen und Links;

rm –**r** rekursives Löschen des Verzeichnisses mit allen Unterverzeichnissen und den enthaltenen Dateien;

rm –**ri** löschen des Verzeichnisse mit allen Unterverzeichnissen und enthaltenen Dateien, mit Rückfrage;

rm –**rf** löschen des Verzeichnisse mit allen Unterverzeichnissen und enthaltenen Dateien, ohne Rückfrage;

rm –**f** löschen ohnen Rückfrage;

Links

ln –**s** ***Verzeichnis linkname*** einen symbolischen Link anlegen; mehrere Namen für ein Verzeichnis; wird das Verzeichnis gelöscht, so hängt der Link in der Luft;

ln (hard link) mehrere Namen für eine Datei; dies ist nur innerhalb eines Dateisystems möglich; die Datei kann erst gelöscht werden, wenn alle auf sie weisenden Links gelöscht wurden;

rm einen symbolischen Link entfernen;

Drucken

cat *Datei*>/dev/lp0 schickt die angegebene Datei direkt unter Umgehung des Druckauftragssteuerungssystems an den Drucker ⇒ brutal;

Dateieigentümer und Dateiattribute

chown *neuer_Eigner Datei(en)* change ownership

chgrp *neue_Gruppe Datei(en)* change group

umask *xyz* ändern der Zugriffsrechte dadurch, daß man von der Standardrechtemaske Rechte abzieht; x,y,z ∈ [0,...7]; 0 entzieht keine Rechte, 7 entzieht alle;
Schema:

x Eigentümerrechte z.B.: x=0 ⇒ user hat alle Rechte

y Gruppenrechte z.B.: x=2 ⇒ group hat keine Schreibrechte

z public-Rechte z.B.: x=7 ⇒ public hat keine Rechte

chmod *xyz datei* nachträgliches Ändern der Zugriffsrechte für eine Datei oder ein Verzeichnis;

- numerische Schreibweise: Die Zahl x steht für die user-Rechte, y für die group-Rechte und z für die public-Rechte. Die 3 Zahlen können, wenn man sie im binär-code anschreibt als 1-aus-n-code-Darstellung der Rechte derjeweiligen Entität betrachtet werden. Die Folge der Rechte ist jeweils read, write und execute. Das execute-Recht bedeutet für Verzeichnisse, daß es betreten werden darf x,y,z ∈ [0,...7]; x=7 ⇒ user hat alle Rechte ;
- symbolische Schreibweise:

  u user

  g group

  o oder **a** public

  \+ ein Recht gewären

  − ein Recht entziehen

  = Gleichsetzen der Rechte verschiedener Entitäten

  r read Leserecht

  w write Schreibrecht

  x execute Ausführungsrecht bzw. Verzeichniszutritt

Set UserID-Bit kurz SUID-Bit: Für den Lauf des Programmes, das das UID-Bit setzt wird der user zum superuser. Nach dem Programmlauf stirbt der Prozeß und alle Rechte sind wieder so wie vorher (z.B.: passwd-Programm).

3.3 Kommunikation

mesg Abfrage ob messages empfangen werden;

mesg yes message-Empfang einschalten;

mesg no message-Empfang ausschalten;

write *Empfänger* dem Empfänger eine Nachricht schreiben;

write *Empfänger Terminal* ist ein user auf mehreren Terminals eingelogged, so muß noch das Terminal angegeben werden;

wall allen eingeloggeden usern als root eine Nachricht senden;

Strg–D Abschließen des Nachrichtentextes;

mail ohne Argument; nachsehen ob mails eingelangt sind;

mail *Empfänger* eine mail schreiben;

'.' mail-Text mit dem Punkt abschließen;

3.4 Zeit und Kalender

date Datum und Uhrzeit ausgeben;

date –s *hh:mm:ss* Zeit einstellen;

cal Ausgabe des Kalenders des aktuellen Monats; mit Jahreszahl als Argument des entsprechenden Jahres;

hwclock Setzen, Anzeiten oder Synchronisieren (mit der Systemzeit) der hardwareclock; nur als root möglich;

at *Uhrzeit* ↩ die hierauf eingegebenen Befehle werden zu der angegebenen Uhrzeit ausgeführt; Abschluß der Befehlssequenz mit *Strg–D*;

atq Anzeige der Abarbeitungsschlange;

atrm *Jobnummer* entfernen dieses Jobs aus der Abarbeitungsschlange;

batch Die hierauf angegebenen Befehle werden ausgeführt, sobald die Systemauslastung unter einen bestimmten Level fällt;

crontab

Der Dämon crond ist für die Abwicklung der Zeitdienste zuständig; er wird automatisch beim Hochfahren gestartet; die Einstellungen stehen in der Datei crontab, die vom Programm crontab bearbeitet wird; nur für superuser; in den Dateien allow und deny steht wer crontab benutzen darf und wer nicht;

crontab –l crontab-Datei betrachten;

3.5 Prozeßmanagement

ps Prozeßliste ausgeben: PID TTY STAT TIME COMMAND

PID Prozeß-ID;

TTY Terminal von dem aus der Prozeß gestartet wurde;

STAT Aktivitätsstatus:

R run

S sleep

TIME Rechenzeit;

COMMAND Aufrufender Befehl;

ps –l Ausgabe in Langform;

ps –ax alle Prozesse, auch die fremden anzeigen;

top ständig aktualisierte Liste der am System laufenden Prozesse;

Prozesse beenden

Programm das im Vordergrund läuft mit eingebauten Kommandos beenden, sonst:

Strg –D bedeutet Dateiende für die Standardeingabe;

Strg –C Kill-Character, Zeichen für Programmbeendigung;

kill *PID* Befehl an den Prozeß in dem das Programm läuft dieses zu beenden, manche Programme ignorieren das aber;

kill –9 *PID* Beendet auch Prozesse die sich wehren;

4 Die Shell

4.1 Dateien die mit der Shell in Zusammenhang stehen

Dateien die Voreinstellungen der shell enthalten:

/etc/profile enthält: Voreinstellungen die für alle Benutzer systemweit gelten, z.B.: Systemzeit, Basissuchpfad in Variable PATH;

.profile benutzerspezifische shell-Einstellungen; liegt im home-Verzeichnis des users;

Bei Verwendung der bash-shell gibt es noch zusätzlich:

bash_profile

bash_login

bash_logout diese Befehle werden beim Beenden der Sitzung ausgeführt;

.bash_history enthält die letzten gemachten Kommandoeingaben;

.inputrc enthält die Tastaturdefinition für den Kommandozeileneditor;

Nochmaliges Ausführen einer versteckten Datei: z.B.: von .profile mit: **. .profile**; .profile ist ein Shellskript das durch Eingabe seines Namens mit vorangestelltem Punkt in der aktiven Shell abgearbeitet wird;

Shellprompt

PS1 Variable des Shellprompts;

echo $PS1 Befehl zur Ausgabe des Shellprompts;

PS2 Variable des Fortsetzungsprompts; erscheint, wenn man z.B. das \-, >> - oder das '(quote)-Zeichen eingibt und Enter drückt, oder wenn eine unvollständige Kontrollstruktur der Shellprobrammierung direkt von der Kommandozeile aus eingegeben wird; zurück zum normalen Prompt mit Strg–C;

IFS Variable die die Trennzeichen der Shell enthält (Leerzeichen, Tab, Zeilenumbruch)

Vordergrund- und Hintergrundausführung

command_1; *command_2*; **...** (mehrere) Kommando(s) im Vordergrund abarbeiten lassen;

command_1 & *command_2* & **...** (mehrere) Kommando(s) im Hintergrund abarbeiten lassen; Shell ist sofort wieder für eine neue Benutzereingabe bereit;

Klammern in der Shell bei Pipes

Befehle *a, b, c*;

$a||b|c$ zuerst wird Befehl *a* gestartet, tritt dabei ein Fehler auf wird die Pipe $b|c$ gestartet;

$\{a||b\}|c$ zuerst wird Befehl *a* gestartet, tritt ein Fehler auf wird Befehl *b* gestartet, die Pipe mit *c* findet in jedem Fall statt;

Exit-Status und die bedingte Kommandoausführung

Exit-Status jedes C- oder Shell-Programm liefert bei fehlerfreier Ausführung den Exit-Status 0 zurück, sonst einen anderen Wert;

$ Exit-Status-Variable;

echo $? Abfrage des Exit-Status;

command_1 **&&** *command_2* läuft *command_1* fehlerfrei, so wird *command_2* gestartet;

command_1 || *command_2* nur wenn bei *command_1* ein Fehler auftritt, wird *command_2* gestartet;

Dateinamen

In der Verzeichnisdatei stehen die Dateinamen mit den dazugehörenden Inodes;

hard link in der gleichen oder in einer anderen Verzeichnisdatei steht ein weiterer Name mit der gleichen Inode;

symbolic link eigener Eintrag mit eigener Inode;

maximale Dateinamenlänge 255 Zeichen;

ls –b nicht darstellbare Dateinamenbestandteile werden durch Ersatzdarstellungen angezeigt;

Wildcards

***** für dieses Zeichen kann jedes beliebige Zeichen beliebig oft, auch nullmal, stehen;

? ein beliebiges Zeichen exakt einmal an dieser Position;

[**...**] ein beliebiges Zeichen, welches in der Klammer angegeben ist, exakt einmal an dieser Position;

[**!...**] ein beliebiges Zeichen, welches **nicht** in der Klammer angegeben ist, exakt einmal an dieser Position;

Aufhebung der Bedeutung von Shell-Sonderzeichen

\ die Bedeutung des nachfolgenden Zeichens wird aufgehoben;

'...' (Shift–#) Text innerhalb der einfachen quotes wird nicht von der Shell interpretiert;

`...` (Akzent) Text innerhalb der einfachen quotes wird nicht von der Shell interpretiert;

Synonyme für Befehle

alias *neuer_befehlsname=befehl* Zuweisung des neuen Befehlsnamens an den Befehl; ist der Befehl mehr als ein Wort (Befehl mit Optionen), so muß man diese Worte unter Anführungsstriche setzen;

unalias *neuer_befehlsaname* Aufhebung der Zuweisung;

alias (ohne Argument) Ausgabe der aktuellen alias-Liste;

Mit den alias' können keine Parameter übergeben werden; ist das letzte Zeichen eines Ersetzungstextes eines alias das Leerzeichen, so wird ein nachfolgendes alias ebenfalls umgewandelt; z.B.:
alias f="echo "; alias g="HALLO"; alias h="echo";
f g ⇒ HALLO wird ausgegeben; h g ⇒ g wird ausgegeben;

Komandozeilenpuffer

!! Äquivalent zur ↑–Cursortaste gefolgt von Return;

!–*n* der *n*-Zeilen zurückliegende Befehl wird ausgeführt;

history Ausgabe desKommandopufferfiles;

history *n* Ausgabe der letzten *n*-Kommandozeilen;

HISTFILE Variable die den Namen des history-files angiebt;

!*Zeichenkette* sucht das letzte Kommando das mit der *Zeichenkette* anfing, und führt es aus;

!?*Zeichenkette* sucht das Kommando in dem die *Zeichenkette* enthalten ist und führt es aus;

ˆ*alt*ˆ*neu* ersetzt in der letzten Kommandozeile den *alt*-Text durch den *neu*-Text und führt diesen Befehl aus;

Kommandozeileneditor

Strg–A Cursor springt an den Anfang der aktuellen Kommandozeile;

Strg–E Cursor springt an das Ende der aktuellen Kommandozeile;

Strg–F Cursor springt ein Zeichen nach rechts;

Strg–B Cursor springt ein Zeichen zurück;

Alt–F Cursor springt ein Wort nach rechts;

Alt–B Cursor springt ein Wort zurück;

Strg–L clear screen;

Tab word-completion; Versuch das Kommandowort (Befehl, Datei, user, ...) zu verfollständigen;

Alt–? Ausgabe der Liste der Möglichkeiten für word-completion;

Alt–/ filename-completion;

Alt–~ username-completion;

Alt–$ complete variable;

Alt–@ complete hostname;

Strg–T transpose characters; Vertauschen des Zeichens an der Cursorstelle mit dem Zeichen davor;

Alt–U upcase word; von der Cursorposition bis zum Ende des Wortes Klein- in Großbuchstaben umwandeln;

Alt–L downcase word; von der Cursorposition bis zum Ende des Wortes Groß- in Kleinbuchstaben umwandeln;

Anpassung des Kommandozeileneditors

bind –v Ausgabe der Liste der aktuellen Editorfunktionen mit den Tastaturbelegungen;

bind –l Ausgabe der möglichen Kommandos;

bind –d Ausgabe der aktuellen Editorfunktionen in der Art, daß diese Ausgabe durch Umleitung in eine Datei und Nachbearbeitung zur Neubelegung verwendet werden kann.

bind –f *Dateiname* Neubelegung der Editorfunktionen gemäß dieser Datei; nichtanzeigbare Zeichen werden folgendermaßen dargestellt:

Strg–Taste \C

Alt–Taste \e

Entfernen–Taste DEL

Leertaste SPACE oder SPC

Enter–Taste RETURN, RET, LFD oder NEWLINE

Tabulator–Taste TAB

bind –q *Kommandoname* Ausgabe der entsprechenden Tastenkombination;

.inputrc Datei die die selbstgewählte Tastaturbelegung enthält (falls nicht die Standardbelegung verwendet werden soll); Kommentare stehen hinter #;

4.2 Umlenkung von Datenströmen

Linux betrachtet Dateien als lange zusammenhängende Folgen von Bytes. Der Inhalt ist nicht von Bedeutung ⇒ einheitliche Sicht von Linux auf die Daten.
⇒ Erweiterung des Dateikonzeptes auf Geräte, Resourcen und FIFOs ist dadurch möglich;
⇒ einfache Umleitung der Datenströme;

Prozeß und Terminal

Der Init-Prozeß ist der erste Prozeß der zum Terminal drei Kommunikationslinien-Standardkanäle anlegt. Jeder Unterprozeß und jede Subshell von ihm erbt diese Verbindungen an seinem Anfang.

Standardkanäle:

Standardeingabe (stdin) Kanalnummer 0 Tastatur - Prozeß Verbindung;

Standardausgabe (stdout) Kanalnummer 1 Prozeß - Bildschirm Verbindung;

Standardfehlerkanal (stderr) Kanalnummer 2 Prozeß - Bildschirm Verbindung;

Eingabe- /Ausgabe-Umlenkung

Programm das alle seine Eingaben von der Standardeingabe liest bis das EOF- (End of File-) Zeichen auftritt, und seine Ausgaben auf die Standardausgabe leitet. Filterprogramme sind einfache Spezialisten für einfache Aufgaben, komplexere Aufgaben können durch Datei- oder Kommandoumlenkungen oder Pipes gelöst werden (⇒ Shellprogrammierung, C, awk, ...).

befehl > *datei* Ausgabe statt am Bildschirm in die Datei umgeleitet; existiert diese Datei schon, so wird sie überschrieben, außer die Shellvariable **noclobber** ist gesetzt;

befehl >> *datei* Ausgabe statt am Bildschirm an die existierende Datei angehängt;

befehl **2>** *datei* Ausgabe des Fehlerkanals (Kanalnummer 2) in die Datei umgelenkt

befehl **2>>** *datei* Ausgabe des Fehlerkanals (Kanalnummer 2) an die Datei angehängt;

befehl > *datei* **2>&1** Umleitung der Standardausgabe und des Fehlerkanals;

befehl > *datei* **2&>1** alternative Schreibweise;

befehl > *datei* **2>&1** | **mail** *mail_adr* Umlenkung der Standardausgabe und des Fehlerkanals in ein mail;

befehl **2>** *fehlerdatei* > *ergebnisdatei* Umlenkung des Fehlerkanals und der Standardausgabe in eine jeweils eigene Datei;

befehl < *datei* Eingabe von der Datei anstatt von der Tastatur;

befehl << *Zeichen* Here-Dokument; alle nachfolgenden Eingabezeilen oder Zeilen des Shellskripts bis zum Aufhebungszeichen *"Zeichen"* sind Eingabe; damit kann ein interaktives Programm mit seinen sonst zwischendurch erforderlichen Eingaben gestartet werden;

befehl n< *datei* Kanal Nr. *n* zum Einlesen öffnen; $n \in [1,2, ...]$;

befehl n> *datei* Kanal Nr. *n* zum Auslesen öffnen;

befehl n<> *datei* Kanal Nr. *n* zum Lesen und Schreiben öffnen;

befehl n<& – Kanal Nr. *n* schließen;

befehl n>& – Kanal Nr. *n* schließen;

Piping

Es gibt 2 Formen des Pipings:

command_1|*command_2*| **...** einfache Pipe; die Kommandos werden von der gleichen Kommandozeile losgeschickt;

command_3 <(*command_1*) <(*command_2*) das *command_3* bekommt die Ausgaben der Kommandos 1 und 2 nicht über eine Datei sondern ein Verzeichnis; die sendenden Kommandos machen einen FIFO-Eintrag ins Verzeichnis /tmp und dieser Verzeichniseintrag wird quasi wie ein Dateiname an das *command_3* geliefert; z.B.: wc <(cat *datei_1*) <(cat *datei_2*)

Kommandoumlenkung

command_1‘*command_2*‘ (grave-Akzent) oder:

command_1$(*command_2*) die Ausgabe des zweiten Kommandos wird zur Eingabe des ersten; z.B.:
aktpfad=$(pwd)
PS1="\u@\h: ‘pwd‘ > "
mail ‘who|cut –d" " –f1‘< *mailtext* damit wird den usern die eingelogged sind ein mail gesendet;

4.3 Prozeßsteuerung

Schickt man ein normalerweise von der Standardeingabe lesendes Programm in den Hintergrund (und macht keine Eingabeumleitung), so gibt die Shell /dev/null als Datenquelle an.
⇒ Das Programm bekommt ein EOF als Eingabe.
⇒ Das Programm beendet sich normalerweise.

Die Ausgabe eines im Hintergrund laufenden Programmes läuft auch (so man sie nicht umleitet) auf das Terminal.

nice –*wert command* Änderung der Priorität des Kommandos; nur der superuser kann die Priorität erhöhen;

nohup *command* **$** verhindert, daß ein lange andauernder Prozeß, der ja von der Kommandoshell, von der er gestartet wurde, abstammt, beim Ausloggen automatisch beendet wird. Der Prozeß wird vom Init-Prozeß adoptiert.

nohup.out diese Datei sammelt alle zwischenzeitlichen Ausgaben der im Hintergrund laufenden Prozesse.

jobs gibt die aktuell laufenden Jobs aus;
[Job_Nr] Symbol Status Befehl

Symbol + oder – für den aktuellen bzw. den Vorgängerjob;

Status Running, Stopped oder Done;

jobs –l [Job_Nr] Symbol PID Status Befehl

Strg–Z Anhalten eines im Vordergrund laufenden Jobs;

bg *Job_Nr* Job in den Hintergrund schicken;

fg *Job_Nr* Job in den Vordergrund schicken;

4.4 Shellvariablen

Shellvariablen werden durch ihre Benutzung ins Leben gerufen; sie müssen nicht vorher deklariert werden wie z.B.: Variablen in C-Programmen;

set –u verhindert, daß auf eine noch nicht definierte Variable zugegriffen wird; z.B.: es ist noch keine Variable *a* definiert;
echo $a ⇒ Ausgabe: leere Zeile
set –u
echo $a ⇒ Ausgabe: a: unbound variable

declare –i *variable* Deklaration der Variable als integer; da die Variable in 32 Bit mit einem Vorzeichenbit dargestellt wird ist der Zahlenbereich $[-2^{31}...2^{31}]$ ca. $2 * 10^9$;

Variablennamen müssen mit einem Buchstaben oder underline beginnen;

declare –r *variable* read only;

declare –x *variable* export; Variable wird auch an subshells weitergegeben;

declare (ohne Argument) Ausgabe der aktuellen Einstellungen der Variablen;

Zuweisungen

variable=*wert* Achtung: keine Leerzeichen einfügen!

variable=**$***variable_2* Zuweisung des Wertes der Variable 2 an die Variable 1;

variable=‘*command*‘ Variable erhält die Ausgabe des Kommandos;

variable=**$(***command***)** Variable erhält die Ausgabe des Kommandos;

read *variable* interaktive Wertzuweisung in Shellskripts;

Ausgabe mit echo

echo *z.B.: Dateinamenmuster* Ausgabe der Dateien die diesem Muster entsprechen; praktisch zur Kontrolle, damit man nicht gleich Befehle auf Dateien anwendet, die man garnicht beglücken will;

echo *text* der Text wird von der Shell analysiert und dann dem Befehl echo als Argument zur Verfügung gestellt;

echo **”***text***”** nur noch Kommandoumlenkung und Variablenzugriff möglich;

echo ´*text*´ Shell analysiert nichts mehr;

z.B.: a=5; echo $a ”$a” \$a ’$a’
Ausgabe: 5 5 $a $a

Steuerzeichen für Ausgabe mit **echo–e**:
(zur Vermeidung von Problemen unter einfache oder doppelte Anführungszeichen setzen)

\b backspace

\c Zeilenvorschubunterdrückung

\f Seitenvorschub

\n Zeilenvorschub

\r carriage return

\t Tabulator

\\ Das Zeichen "\"

\nnn das ASCII-Zeichen, dessen Oktalnummer *nnn* ist

\a Alarm

Verkettung:
$a=5;
echo $a,Leute ⇒ 5,Leute
echo $a Leute ⇒ 5 Leute
echo ${a}Leute ⇒ 5Leute

Shellvariablen und Kommandoaufrufe

Shellvariablen können in Kommandozeilen verwendet werden, da ihre Inhalte interpretiert werden. Von Bedeutung nur in Shellskripts, da bei interaktiver Kommandoeingabe Kommandosynonyme zur Verfügung stehen.

Da die Shell nur 1 Analyse der Kommmandozeile durchführt werden Kommandozeilen, deren Shellvariablen selbst Shellsonderzeichen und Variablenzugriffe enthalten, meist fehlerhaft verarbeitet.
Z.B.: pipe="| wc"
ls $pipe ⇒ Ausgabe: no such file or directory (und nicht die Anzahl der Wörter und Buchstaben des ls-Kommandos.

eval *ausdruck* der Ausdruck wird von eval genau so analysiert wie es die Shell macht; eval kann auch geschachtelt werden; die Shell analysiert dann das Ergebnis noch einmal;
eval ls $pipe ⇒ Ausgabe der Anzahl der Wörter und Buchstaben;

Gültigkeitsbereich von Variablen

Jeder Prozeß trägt einige Daten mit sich herum, die als seine Umgebung (environment) bezeichnet werden. Z.B.: enthält die Umgebung der Shell die geöffneten Standardkanäle, Datenzugriffe und das aktuelle Arbeitsverzeichnis.

lokale und globale Variablen

declare –**x** *variable* (x...export), oder:

export *variable* die so deklarierten Variablen werden an eine Subshell weitergegeben;

declare –**f** *shell_fkt.* oder:

export –**f** *shell_fkt.* die so deklarierten Shellfunktionen werden an eine Subshell weitergegeben;

Veränderungen einer Variablen in der Subshell werden nicht an die aufrufende Shell weitergegeben.

Löschen von Variablen

variable=↩ Gleichsetzen der Variable mit der leeren Zeichenkette;

unset *variable* die Variable als ganzes löschen; dies funktioniert nicht, wenn die Variable noch den Read-Only-Status hat (vorher den Read-Only-Status beenden mit: readonly –n);

4.5 Berechnungen mit der Shell

Zahlenbereich $[-2^{31}...2^{31}]$ ca.$2*10^9$; 32 Bit, 1 Vorzeichenbit; keine Warnung bei Zahlenbereichsüberschreitung;

echo $[*Ausdruck*] ⇒ Ausgabe des Rechenergebnisses;
z.B.: a=5; b=2
c=$[$a+$b]; echo c ⇒ Ausgabe: 7;

Operatoren

!, ~ logische, bzw. bitweise Negation;

*****, /, % Multiplikation, Division, Modulo (Rest der ganzzaligen Division);

+, - Addition, Subtraktion;

<<, >> Verschieben des Bitmusters nach links bzw. rechts; $a >> b$ schiebt das Bitmuster von a um b-Stellen nach rechts;

<=,>= kleiner gleich, größer gleich; das Ergebnis liefert 1 für TRUE oder 0 für FALSE;

==, != Gleichheit bzw. Ungleichheit;

&, | bitweises UND bzw. ODER;

$$, || logisches UND bzw. ODER;

=*operator*= Zuweisung und kombinierte Zuweisung; als Operator sind alle obigen 2-Operanden-Operationen erlaubt;

4.6 Das Kommando set

Mit **set** kann das Verhalten der Shell verändert werden im Gegensatz zu **stty**, das das Verhalten des Terminals verändert. Da man mit der Shell nur mittels des Terminals kommuniziert sind komplizierte Verwicklungen möglich.

5 Linux-Werkzeuge

5.1 Reguläre Ausdrücke

Metazeichen die von vielen Programmen interpretiert werden:

[^...] Verneinung der Zeichenklasse, wie [! ...];

^ Zeilenanfang;

$ Zeilenende;

< Wortbeginn;

> Wortende;

(...) Gruppen von Ausdrücken;

\ Ausblendung der Sonderbedeutung des folgenden Zeichens;

Von manchen Programmen interpretierte Metazeichen:

+ das vorangehende Zeichen 1-mal oder öfter;

{*n*,*m*} ein Wiederholungsintervall;

{*n*} ein genauer Wiederholungswert;

a|*b* alternative Ausdrücke;

find

find [Verzeichnis][–Optionen][–Test][–Aktionen]

Verzeichnis

find sucht in dem angegebenen Verzeichnis und seinen Unterverzeichnissen; dieser Eintrag darf nicht ausgelassen werden, mindestens: /

Optionen

–**daystart** mißt die Zeitmarken der Dateien vom Beginn des heutigen Tages;

–**depth** zuerst dieVerzeichnisinhalte, dann die Verzeichnisse selbst bearbeiten;

–**maxdepth** *N* höchstens *N*-Stufen im Verzeichnisbaum nach unten gehen;

–**mindepth** *N* mindestens *N*-Stufen im Verzeichnisbaum nach unten gehen;

–**xdev** Verzeichnisse in anderen Dateisystemen als dem aktuellen werden nicht durchsucht;

–**follow** folgt symbolic links;

–**help**

Test

"*suchtext***"** Suche nach Namen (des users, der group, Dateinamen, ...);

zahlenwert Suche nach dem exakten Zahlenwert;

+zahlenwert Suche nach größerer als der angegebenen Zahl;

–zahlenwert Suche nach kleinerer als der angegebenen Zahl;

–**amin** *N* Zugriff vor *N*-Minuten;

–**anewer** *file* auf diese Datei(en) wurde nach dem Zugriff auf *file* zugegriffen;

–**cmin** *N* Statusänderung der Datei vor *N*-Minuten;

–**cnewer** *file* die Statusänderung wurde nach der Statusänderung von *file* durchgeführt;

–**ctime** *N* die Statusänderung wurde vor *N*-Tagen vorgenommen;

–**mmin** *N* der Dateiinhalt wurde vor *N*-Minuten geändert;

–**mtime** *N* der Dateiinhalt wurde vor *N*-Tagen geändert;

–**newer** *file* Veränderung erfolgte nach der letzten von *file*;

–**used** *N* Zugriff inerhalb von *N*-Tagen nach der letzten Änderung;

Aktionen

–**exec** *command* nachdem eine Datei gefunden wurde wird mit ihr automatisch das Kommando durchgeführt;

–**ok** *command* nachdem eine Datei gefunden wurde wird mit ihr nach Rückfrage das Kommando ausgeführt;

grep

fgrep (fast grep) ist schneller als **grep**, kann aber weniger;

egrep (extended grep) ist langsamer als **grep**, kann aber mehr;

grep [–[[A B]]< *num* >][–[CEFGVchilnqsvwx]][–[ef]]< *expr* >[< *files* >]

A *n* gibt *n* Zeilen nach der Fundstelle aus;

B *n* gibt *n* Zeilen vor der Fundstelle aus;
z.B.: grep –A2 –B5 '*suchtext*' *datei*
gibt gibt die gefundenen Zeilen mit den jeweils 2 vorhergehenden und jeweils 5 nachfolgenden Zeilen aus; die einzelnen Fünde werden durch '– –' getrennt;

–*n* gibt *n* Zeilen vor und nach dem Fund aus;

–b Ausgabe mit der Positionsangabe;

–i Groß-/Kleinschreibung ignorieren;

–l Nur die Namen der Dateien mit Fünden ausgeben;

–n Ausgabe mit Zeilennummern;

–s nur die Fehlermeldungen ausgeben;

–v nur die Zeilen ohne den Suchtext ausgeben;

–w nur die Zeilen ausgeben, in denen der Suchtext als ganzes Wort vorkommt;

–x nur die Zeilen ausgeben, in denen der Suchtext als ganze Zeile vorkommt;

5.2 file

file *datei* gibt an welchen Typs die Datei ist; mögliche Ausgaben: ascii text, shell script, Linux/i386, symbolic link, block special; ...

cat

cat (concatenate) hängt die angegebenen Dateien zusammen; für kurze Dateien ist es als schneller viewer geeignet;

–b alle nicht leeren Zeilen nummerieren;

–n alle Zeilen nummerieren;

–s mehrere aufeinanderfolgende Leerzeilen werden zu 1 Leerzeile umgewandelt;

–E Anzeige von "$" an jedem Zeilenende;

–T Tabs als "^I" darstellen;

tac

Wie cat nur die Zeilenreihenfolge ist umgekehrt;

more und less

less ist umfangreicher und muß nicht erst die ganze Datei einlesen bevor die Ausgabe beginnt ⇒ schneller;

less –? Anzeige der Flags d.h. der internen Kommandos;

tail

gibt die letzten Zeilen (standardmäßig 10) einer oder mehrerer Datei(en) aus;

head

gibt die ersten Zeilen (standardmäßig 10) einer oder mehrerer Datei(en) aus;

sort

sort +POS1 –POS2 sortiert gemäß der Zeichen von POS1 bis POS2; POSx kann in Tabellen als *feld.zeichen* festgelegt werden;

–**b** führende Leerzeichenunbeachtet lassen;

–**d** nur Zeichen a–z, A–Z und 0–9 berücksichtigen;

–**f** Groß- und Kleinschreibung nicht getrennt behandeln;

–**r** (reverse) umgekehrte Sortiereihenfolge;

uniq

Auswahl der Zeilen einer Datei jenachdem ob sie einmal oder öfter enthalten sind;

–**u** unterdrückt mehrfache Zeilen;

–**d** gibt nur die mehrfach enthaltenen Zeilen aus;

–**c** (count) Ausgabe der Zeilen mit ihrer Anzahl des Auftretens vorangestellt;

tee

T-Stück in einer Pipe; Abzweigung eines Zwischenergebnisses in eine Datei;
...| **tee** *datei* | ...

od

(octal dump) Betrachten von binären Dateien;

–**Ad** mit dezimaler Byteposition;

–**Ao** mit oktaler Byteposition;

–**Ax** mit hexadezimaler Byteposition;

–**An** ohne Byteposition;

–**a** Datenausgabe als ASCII-Zeichen;

–**x** Datenausgabe als ASCII-Zeichen;

–**f** Datenausgabe als Fließkommazahl;

cut

Ausschneiden von Spalten aus einer Textdatei;

–**c** *a*–*b* schneidet die Bites *a* bis *b* aus der Datei aus;

–**f** *n* schneidet die *n*-te Spalte aus; Spaltendelimitor (-Trennzeichen) ist der Tab;

–**d"***x***"** Zeichen *x* als Spaltendelimitor definieren; tritt das gewählte Zeichen mehrfach hintereinander auf so sieht cut leere Felder ⇒ Abhilfe mit **tr**;

fold

Begrenzung der Zeilenlänge durch Umbruch;

split und csplit

Zerlegen von Dateien; bei csplit kann man ein Suchmuster angeben, wobei die Zerlegung so erfolgt, daß die Zeile die das Suchmuster enthält zur 1. Zeile der nächsten Splitterdatei wird. Das Suchmuster wird nach dem Dateinamen angegeben. In geschweiften Klammern kann angegeben werden, wie oft das Suchmuster angewendet werden soll, keine Angabe ⇒ nur 1 mal; {*} ⇒ sooft wie möglich;

join

Zusammenfügen mehrerer sortierter Dateien in Abhängigkeit von einer jeweils anzugebenden Schlüsselspalte;

paste

Zeilenweises Zusammenfügen von Dateien; alle jeweils 1. Zeilen werden zu einer neuen Gesamt-1.-Zeile (die Bruchstücke sind durch Tabs getrennt);

Vergleiche

cmp

Byteweiser Vergleich zweier Dateien; bei Ungleichheit wird abgebrochen und der Exit-Status ist 1;

–**verbose** es wird eine Liste der Unterschiede ausgegeben;

comm

Vergleich von zwei sortierten Dateien;
Ausgabe ist 3-spaltig:

1. Spalte: Einträge die nur in der 1. Datei stehen;

2. Spalte: Einträge die nur in der 2. Datei stehen;

3. Spalte: Einträge die in beiden Dateien enthalten sind;

5.3 Umwandlungen

expand

Alle Tabs in eine gewünschte Anzahl von Leerzeichen umwandeln;

tr

Zeichen ersetzen oder beseitigen; funktioniert nur innerhalb einer pipe und jeweils nur eine Ersetzung; z.B.:
cat *datei* | tr "[:punct:]" " " | tr –s " " | less
in der angegebenen Datei werden alle Punktationszeichen durch blanks ersetzt, und anschließend blank-Ketten durch einzelne blanks ersetzt;
tr [OPTIONS] SET1 [SET2]

SET1 Liste der zu ersetzenden Zeichen;

SET2 Liste der Ersetzungszeichen; n-tes Element ersetzt das n-te Element von SET1;
OPTIONS:

–**c** (complement) Ersetzung des Gegenteils der in SET1 angegebenen Zeichen durch Zeichen von SET2;

–**s** (squeeze) mehrfach vorkommende Zeichen werden durch ein einzelnes Zeichen ersetzt;

trs Filterprogramm zum Ersetzen von strings

5.4 Formatierungen

nl

nl [OPTION] ... [FILE]
Textdateien (oder wenn keine Datei angegeben wird die Standardeingabe) für einfachen Ausdruck formatieren; Ausdruck mit Kopfzeile, Fußzeile, Seitennummer, Zeilennummerierung, ...

Kopfteil wird mit "\ : \ : \ :" eingeleitet;

Körper wird mit "\ : \ :" eingeleitet;

Fußteil wird mit "\ :" eingeleitet;

pr

pr [OPTION] ... [FILE] ...
Ausdruck eines Textfiles (oder wenn keine Datei angegeben wird der Standardeingabe); in der Kopfzeile stehen standardmäßig: Dateiname, Seitennummer und aktuelles Datum;

6 Editoren des Linux-Systems

6.1 ed

Zeilenorientierter Editor, d.h. man kann nur ganze Zeilen löschen, ersetzen ...;

ed *datei* wird kein Dateiname oder ein neuer angegeben, so generiert man eine neue Datei;

Ausgabe: Anzahl der gelesenen Bytes; ed ist ursprünglich im command-modus;

Command-Modus-Befehle: q (quit) Beenden von ed;

w (write) Speichern der Datei;

l (last) Anzeigen der letzten Zeile; nichtdruckbare Zeichen werden durch Ersatzdarstellungen angezeigt;

p (print) Anzeige der letzten Zeile so wie sie im Druck erscheint;

n Zeile mit vorangestellter Zeilennummer;

l-, p- oder n-Befehl mit vorangestellter Zeilennummer zeigt die angegebene Zeile an;
Anzeige eines Zeilenbereiches: *a, b X*

a Anfangszeilennummer;

b Endzeilennummer; für die letzte Zeile der Datei steht: $;

X Befehl l, p oder n;

Texteinfügen

a return (append) Text hinter der aktuellen Zeile einfügen;

i return (insert) Text vor der aktuellen Zeile einfügen;

dann die neue Zeile eingeben und die Eingabe mit return abschließen; mit Punkt gefolgt von return springt ed wieder in den command-modus;

Text löschen

d return (delete) Aktuelle Zeile löschen;

*n***d return** (delete) Zeile mit Nummer *n* löschen;

Kopieren

a **t** *b* Zeile *a* wird hinter Zeile *b* kopiert; für die letzte Zeile der Datei steht: $;

Relative Zeilenadresse

+*n* *n*-Zeilen nach oben;

-*n* *n*-Zeilen nach unten;

7 Shellprogrammierung

7.1 Allgemeines

Die folgenden Erläuterungen beziehen sich auf die bash-Shell; in der Korn-Shell gelten sie größtenteils auch; die Shellprogrammierung in der C-Shell ist aber völlig inkompatibel dazu;

Elemente der Shellprogrammierung:

Shellbestandteile
- interne Kommandos der Shell
- Kontrollstrukturen der Shellsprache
- Shellvariablen

Programme und Shellskripts mitgeliefert, selbstentwickelt, ...

Kommentar

alles in einer Zeile hinter der Raute stehende gilt als Kommentar;

Wohin mit den selbstgeschriebenen Shellskripts?

Nich in \bin oder \usr\bin wegen organisatorischer Probleme bei updates, sondern:
eigene in ein \bin subdirectory unter dem eigenen home-Verzeichnis und systemweite (öffentliche) unter \usr\local\bin
PATH-Varaiable entsprechend setzen!

7.2 Ausführungsformen

expliziter Aufruf einer Subshell

bash *skript* Ausführung in einer explizit aufgerufenen Subshell, die nur für die Laufzeit der Shell existiert; Vorteile des expliziten Aufrufes:

- Sicherheit;
- auch möglich, wenn das Skript für eine andere als die aufrufende Shell geschrieben wurde;
- nur das Leserecht muß gesetzt sein;

Impliziter Aufruf einer Subshell

skript Das Shellskript läuft in einer eigenen implizit aufgerufenen Subshell;
Vorteile:

- Sicherheit;
- Lese- und Ausführungsrechte müssen gesetzt sein;
- Aufruf wie der von Linux-Befehlen oder anderer externer Programme;

Ausführung in der aktiven Shell

. *skript* Das Shellskript wird in der aktiven, d.h. in der aufrufenden Shell ausgeführt;

Befehl exit gute Shellskripts enthalten diesen Befehl meistens; so kann nach dem Ablauf des Shellskripts kontrolliert werden ob der Ablauf ohne Probleme verlief; der Befehl exit beendet aber auch die Shell in der das Shellskript läuft! - bei Ausführung in der aktiven Shell meldet man sich somit ab.

Probleme

Beginnt ein Shellskript mit:

#!\bin\bash so läuft das Shellskript automatisch in der bash-Shell ab;

#!\bin\csh so läuft das Shellskript automatisch in der C-Shell ab;

Befehl exec exec ruft ein echtes Binärprogramm ohne eigener Subshell auf (Vorteil: schnell und geringe Systembelastung); alle weiteren Befehle im Shellskript werden ignoriert!

7.3 Daten-Ein- und Ausgabe

Argumentevariablen

Sie werden Beim Aufruf des Shellskripts als Argumente bzw. Parameter übergeben.

${**$n$}** Variable die das n-te Argument enthält; $n \in [1...9]$ bei der Bourne-Shell
mit dem Befehl **shift** lassen sich die dahinterliegenden Variablen holen
$n \in [1...\infty]$ bei der Bourne-Shell

$0 enthält den Namen des Shellskripts

$# enthält die Anzahl der übergebenen Argumente

$@** oder **$* enthalten jeweils die ganze Argumenteliste

Interaktive Shellskripts

read *a* die Variable *a* wird interaktiv eingelesen; es wird alles bis zum abschließenden return eingelesen;

read *a b c* die 3 Variablen *a, b* und *c* werden eingelesen; jede Variable erhält ein Wort, falls Worte überzählig sind werden sie auch der letzten Variable zugewiesen; die möglichen Trennzeichen zwischen den Variablen stehen in der Shellvariablen IFS;

Ausgabe

echo "*Ausgabetext***"** der Ausgabekanal kann festgelegt werden, ansonsten Standardausgabe;

Andere Verfahren der Dateneingabe

Sie sind anzuwenden wenn die Daten immer die gleichen sind, oder nicht vom Benutzer des Skripts bereitgestellt werden.

Variablenbelegung vor dem Skriptaufruf Läuft das Skript in einer eigenen subshell so müssen die Variablen in der aufrufenden shell mit **export** vererbbar gemacht worden sein. Durch Argumenteerweiterungen könne Probleme die dur Nichtbelegung der erforderlichen Variablen entstehen können vermieden werden.

Dateiumleitung und Piping in Shellskripts Umleitungen in oder aus anderen Dateien oder Programmen sind möglich. Z.B.: Ausgabe in den Fehlerkanal:
echo "*text*" >&2
exit 1
Zeilenweise Ausgabe mit Zeilennumerierung der im Aufruf spezifizeirten Datei:
cat $1 | while read zeile
do
echo -e "$zno \t: $zeile"
let zno+=1
done > $2
exit 0

Kommandoumlenkung

8 gawk

8.1 Allgemeines

awk ist ein von Aho, Weinberger und Kernighan entwickelter Reportgenerator. Ein Reportgenerator ist dazu da Dateien zeilenweise zu lesen, die Zeilen in Felder zu zerlegen und je nach Aufgabe daraus die entsprechenden Antworten zu erzeugen (Bericht aus einer Datenbank). gawk ist die Gnu-Version.

8.2 Aufruf des gawk

Wird beim Start von awk oder gawk keine Datei angegeben, so wird die Standardeingabe bis zum Abschluß mit Strg–D bearbeitet.

gawk *'program' datei* oder

gawk –**f** *programmdatei datei* Das Programm kann in der Kommandozeile direkt angegeben werden oder in der Programmdatei stehen; Programm in einfache Anführungsstriche setzen!; die Datei enthält die Datensätze;

Optionen

–**F** *fs* Angabe eines fieldseperators *fs*; damit wird die Variable *FS* neu belegt;

–**v** *var* Belegung der Variablen *var* mit einem Wert;

–**f** *datei* Angabe des Programms in der Programmdatei;

–**W** Gnu-awk spezifische Optionen;

–**W compat** Gnu-awk läuft kompatibel zum original UNIX-awk;

–**W lint** Test auf semantisch zweifelhafte oder schlecht portable Konstrukte;

–**W posix** POSIX-kompatibler Modus;

\x Escape-Sequenzen werden nicht erkannt;

func als Synonym für **function** wird nicht erkannt;

**** und ****= können nicht anstelle der ^**and** und ^= Operatoren verwendet werden;

8.3 Aufruf in Shellskripten

mit interaktivem Dateinamen

```
#!/bin/bash
echo –e “Dateiname: \c”
read dateiname
gawk ‘{print }’ $dateiname
```

todschicke Variante

#!/usr/bin/gawk –f
alle nachfolgenden Zeilen des Shellskripts werden als gawk-Programm abgearbeitet;

8.4 Aufbau von gawk-Programmen

Ein gawk-Programm besteht aus mindestens einer Zeile die enthält:
Muster { *Aktion(en)* }
das / die Muster werden in jeder Eingabe- /Dateizeile einmal gesucht und falls vorhanden wird / werden die Aktion(en) ausgeführt; wird kein Muster angegeben werden die Aktionen auf jeden Fall ausgeführt;

Einfaches gawk-Programm

gawk '{print "Hello, world"}'
nach jeder beliebigen weiteren Eingabe wird "Hello, world" geantwortet, bis man die Eingabe von der Standardeingabe mit Strg–D abschließt;

Papagei-Programm

gawk '{print}'
die Eingabe wird auf jeden Fall zeilenweise "nachgeplappert";
gawk '{print $0}'
liefert das gleiche Resultat;

Variablen

$0 diese Variable enthält die ganze Eingabezeilevon der Standardeingabe, oder die aktuelle Zeile der Datei; daher haben obige zwei Beispiele dieselbe Wirkung;

$*n* enthält das *n*-te Feld (Wort) der aktuellen Zeile; $n \in [1 \ldots 255]$;

$NF enthält das letzte Wort der Zeile;

NF (number of fields) Anzahl der Worte der aktuellen Zeile;

NR (line number) Zeilennummer;

OFS (output field seperator) enthält das Feldtrennzeichen für die Ausgabe;

gawk '{print "Insgesamt", NF, "Worte, das letzte ist: ",$NF}'
gibt die Anzahl der Wörter in der Zeile und das letzte Wort aus;

Formatierte Ausgabe

printf (print formatted) gibt die Variablen in der gemäß der Formatanweisung entsprechenden Weise aus; jede Formatanweisung beginnt mit einem %
printf (*formatanweisung, Variable(n)*
z.B.:
gawk '{printf("%d\t%d\n", $1,$2)}'
gibt die Variablen $1 und $2 getrennt durch einen Tab aus und macht einen Zeilenvorschub;

Typzeichen:

d die Variable wird als dezimale Ganzzahl dargestellt;

i dezimale Ganzzahl;

u (unsigned) Ganzzahl ohne Vorzeichen;

o (otale) Ganzzahl;

x hexadezimale Ganzzahl, 10 bis 15 werden durch a bis f dargestellt;

X hexadezimale Ganzzahl, 10 bis 15 werden durch A bis F dargestellt;

f Vorzeichenbehaftete Darstellung der Form [-]dddd.dddd

c einzelnes Zeichen;

s Zeichenkette;

Ersatzdarstellungen für nicht druckbare Zeichen:

\\ der backslash;

a das Alarmsignal (beep);

b backspace;

r Wagenvorlauf;

f (formfeed) Seitenvorschub;

t Tabulatorschritt;

v vertikaler Vorschub;

n Zeilenvorschub;

c Unterdrücken des Zeilenvorschubes;

8.5 Muster von gawk-Programmen

Vergleichsausdrücke

erfolgt der Vergleich nur zwischen Zahlen, so wird er numerisch durchgeführt, ansonsten werden Zeichenketten verglichen; z.B.: *"Hamburg"* < *"München"* stimmt, da der ASCII-Code von H kleiner ist als der von M;

< kleiner als;

<= größer gleich als;

== Gleichheit;

!= Ungleichheit;

>= größer gleich;

> größer;

~ Mustervergleich;

!~ verneinter Mustervergleich;

z.B.:
gawk '$0 >= 22'
plappert jede Zeichenkette, deren Zahlenwert größer als 22 ist nach;
gawk '$0 >= 22 {print}'
macht genau das gleiche;
gawk '/Anne Hektor/'
plappert alle Zeilen in denen der Name steht nach;
gawk '$0 "Anne Hektor"{print}'
macht genau das gleiche;

/zeichenkettenmuster/ das zwischen den slashes stehende Muster; z.B.: /Aaanne Hektor/